Javascript avancé
Module 3 – Javascript et les API

OBJECTIFS

Savoir ce qu'est une API et comment cela fonctionne

Connaitre plus particulièrement le fonctionnement des API RestFul

Savoir comment envoyer des requêtes asynchrones

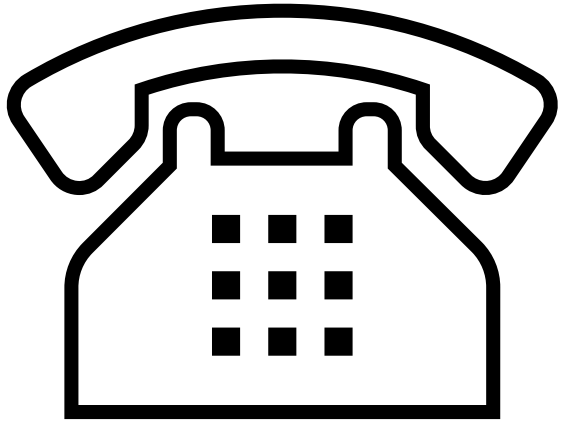
SOMMAIRE

C’est quoi une API ?

Interface

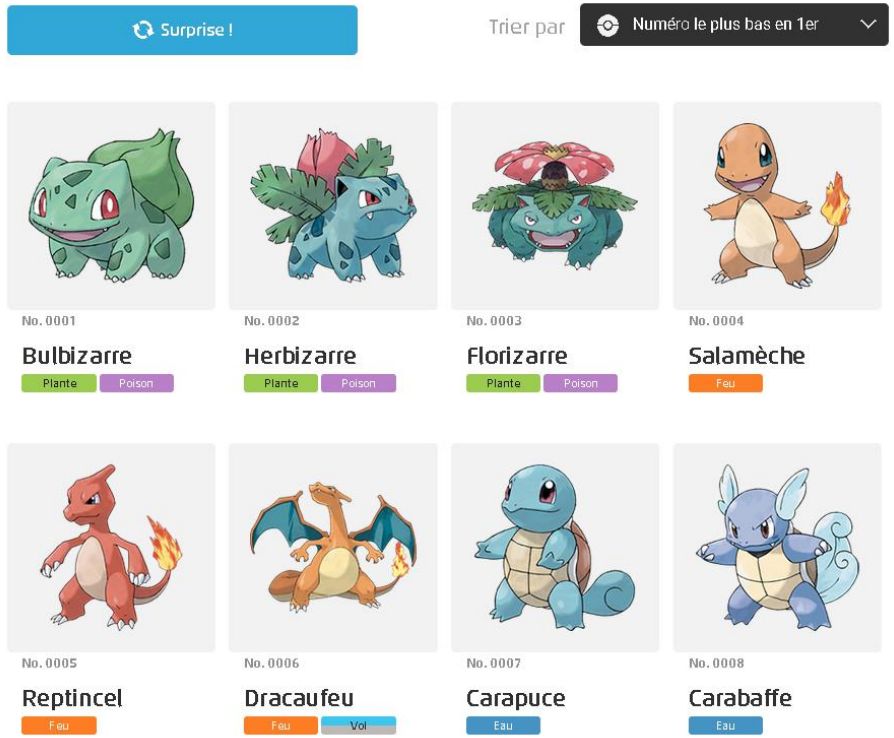
Une interface est une abstraction du fonctionnement interne d'un objet.

Toutes les interfaces nous fournissent, néanmoins, un moyen de les utiliser.

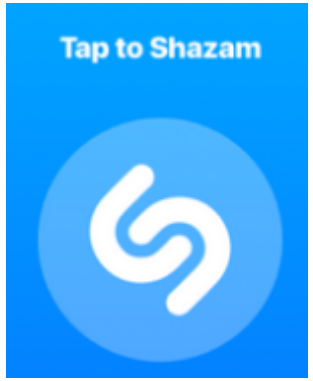


API – un contrat


```
count:      1281
next:       "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20"
previous:   null
results:
  0:
    name:   "bulbasaur"
    url:    "https://pokeapi.co/api/v2/pokemon/1/"
  1:
    name:   "ivysaur"
    url:    "https://pokeapi.co/api/v2/pokemon/2/"
  2:
    name:   "venusaur"
    url:    "https://pokeapi.co/api/v2/pokemon/3/"
```

API web

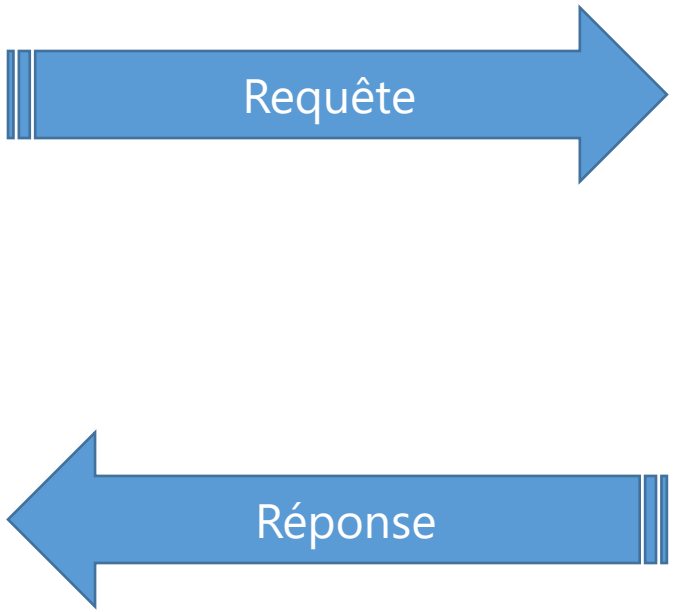


Est-ce que mon téléphone possède, sur son disque dur, toutes les chansons du monde ? ¯_(ツ)_/¯

Alors, comment arrive-t-il à la reconnaitre ? 🤔

HTTP



Uniform Resource Identifier

Une API organise ses données et/ou ses ressources en utilisant un ensemble d'URI

« https://www.mon_api.com/livres »

⚠ Une URI n'est pas une URL

GET

Un client va faire une requête pour accéder aux ressources exposées par l'API

```
GET /livres HTTP/1.1
Host: www.mon_api.com
Accept: application/json
```

Format d'une requête

- Un verbe HTTP
- Une entête
- Un corps

```
POST /livres HTTP/1.1
Host: www.mon_api.com
Accept: application/json
Authorization: <Bearer>
Connection: keep-alive

{
    titre: 'Le rouge et le noir'
}
```

DÉMONSTRATION

Navigateurs Web, Curl et WebStorm

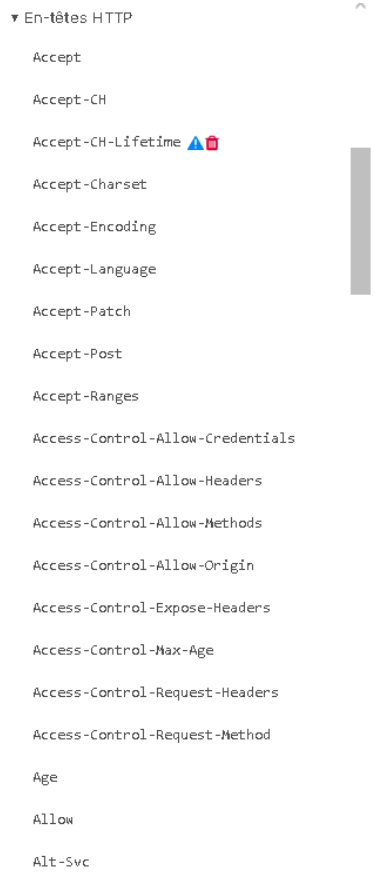
Les requêtes HTTP

Verbes HTTP

GET	Lire
POST	Créer
PATCH / PUT	Mettre à jour
DELETE	Détruire

Headers

https://developer.mozilla.org/fr/docs/Web/HTTP/Headers



En-têtes HTTP

Les en-têtes HTTP permettent au client et au serveur de transmettre des informations supplémentaires avec la requête ou la réponse. Un en-tête de requête est constitué de son nom (insensible à la casse) suivi d'un deux-points `:`, puis de sa valeur (sans saut de ligne). L'espace blanc avant la valeur est ignoré.

Des en-têtes propriétaires personnalisés peuvent être ajoutés en utilisant le préfixe `x-`, mais cette convention a été abandonnée en juin 2012, en raison des inconvénients qu'elle a présenté lorsque des champs non standard sont devenus standard dans RFC 6648; les autres en-têtes possibles sont listés dans une liste IANA et ont été définis dans la RFC 4229. IANA maintient également une liste des propositions de nouveaux entêtes HTTP.

Les en-têtes peuvent être groupés selon leur contexte :

- En-tête général : en-têtes s'appliquant à la fois aux requêtes et aux réponses mais sans rapport avec les données éventuellement transmises dans le corps de la requête ou de la réponse.
- En-tête de requête : en-têtes contenant plus d'informations au sujet de la ressource à aller chercher ou à propos du client lui-même.
- En-tête de réponse : en-têtes contenant des informations additionnelles au sujet de la réponse comme son emplacement, ou au sujet du serveur lui-même (nom et version, etc.)
- En-tête d'entité : en-têtes contenant plus d'informations au sujet du corps de l'entité comme la longueur de son contenu ou son type MIME.

Format d'une réponse

- Un code de statut
- Une entête
- Un corps

```
HTTP/1.1 200 OK

Server: nginx
Connection: keep-alive

{
    "id": 42
    "nom": "konami"
    "code": "hhbbgdgdba"
}
```

Code statut

200-299 : Tout va bien

300-399 : Messages de redirection

400-499 : Erreurs du client

500-599 : Erreurs du serveur

https://http.cat/{code_de_statut}

Ressources

Une ressource est la représentation d'une donnée


```
{
    "name": "Obi-Wan Kenobi",
    "height": "182",
    "mass": "77",
    "hair_color": "auburn, white",
    "skin_color": "fair",
    "eye_color": "blue-gray",
    "birth_year": "57BBY",
    "gender": "male",
    "homeworld": "https://swapi.dev/api/planets/20/",
    "films": [
        "https://swapi.dev/api/films/1/",
        "https://swapi.dev/api/films/2/",
        "https://swapi.dev/api/films/3/",
        "https://swapi.dev/api/films/4/",
        "https://swapi.dev/api/films/5/",
        "https://swapi.dev/api/films/6/"
    ],
    "species": [],
    "vehicles": [
        "https://swapi.dev/api/vehicles/38/"
    ],
    "starships": [
        "https://swapi.dev/api/starships/48/",
        "https://swapi.dev/api/starships/59/",
        "https://swapi.dev/api/starships/64/",
        "https://swapi.dev/api/starships/65/",
        "https://swapi.dev/api/starships/74/"
    ],
    "created": "2014-12-10T16:16:29.192000Z",
    "edited": "2014-12-20T21:17:50.325000Z",
    "url": "https://swapi.dev/api/people/10/"
}
```

JavaScript Object Notation

```
{ } // Un objet
[ ] // Une collection
```

```
{
    "name": "Obi-Wan Kenobi",
    "films": [
        "https://swapi.dev/api/films/1/",
        "https://swapi.dev/api/films/2/",
        "https://swapi.dev/api/films/3/",
        "https://swapi.dev/api/films/4/",
        "https://swapi.dev/api/films/5/",
        "https://swapi.dev/api/films/6/"
    ]
}
```

RESTful

La plupart des API sont dites « RESTful ».
Elles suivent un ensemble de règles ou contraintes nommées :

Representational State Transfert

i RESTful est le standard depuis les années 2000

Contraintes REST

- Client serveur
- Sans état
- Avec une mise en cache ou non
- En couches
- Code à la demande
- Interface uniforme

https://restfulapi.net/

Swagger


Book
GET /books Retrieves the collection of Book resources.
POST /books Creates a Book resource.
GET /books/{id} Retrieves a Book resource.
PUT /books/{id} Replaces the Book resource.
DELETE /books/{id} Removes the Book resource.
PATCH /books/{id} Updates the Book resource.
PUT /books/{id}/generate-cover Replaces the Book resource.

DÉMONSTRATION

Swagger

L'asynchronisme

La boucle d'évènement

```
console.log('évènement 1');
setTimeout(() => console.log('évènement 2'), 0);
Promise.resolve().then(() => console.log('évènement 3'));
console.log('évènement 4');
```

1 4 3 2 😵

Fetch()

fetch() permet de récupérer des ressources à travers le réseau, en utilisant HTTP, et de manière asynchrone.

```
const reponseAPI = fetch('https://api.agify.io/?name=Guy');
console.log(reponseAPI);
```

Promesse

fetch() retourne une promesse

La méthode then() prend en entrée une promesse et retourne une promesse. Elles peuvent donc être « chainées ».

```
console.log('⌚');
reponseAPI
    .then(donnees => donnees.json())
    .then(ressources => {
        console.log('🤗', ressources);
        console.log(`Bonjour ... ${ressources.name}`);
    })
    .catch(erreur => console.error('😡', erreur));
console.log('⏱');
```

DÉMONSTRATION

fetch pokeAPI

async

async permet de « transformer » le retour d'une fonction en promesse.

```
const diner = async (nom) => {
    const choix = {
        'pizza': '🍕',
        'hamburger': '🍔',
        'tacos': '🌮',
        'ramens': '🍜',
        'biere': '🍺',
        'the': '🥤'
    };
    return choix[nom];
}
diner('pizza').then((choix) => console.log(choix));
```

await

await « attend » la résolution d'une promesse avec d'exécuter la suite du code.

```
console.time("Temps d'exécution");
const deuxiemeDiner = async () => {
    const solide = await diner('hamburger');
    const liquide = await diner('the');
    return [
        solide,
        liquide
    ];
}
deuxiemeDiner().then(console.log);
console.timeEnd("Temps d'exécution");
```

DÉMONSTRATION

Promesses, Await, Async

TRAVAUX PRATIQUES

ISS

Conclusion

- Vous connaissez le fonctionnement des API et plus particulièrement celui des API RestFul
- Vous savez utiliser les éléments Javascript permettant de lancer des requêtes asynchrones